LA-UR-13-28342 (Accepted Manuscript)

TADSim: Discrete Event-Based Performance Prediction for Temperature-Accelerated Dynamics

Mniszewski, Susan M.
Junghans, Christoph
Voter, Arthur F.
Perez, Danny
Eidenbenz, Stephan J.

To be published in: ACM Transactions on Modeling and Computer Simulation

DOI to publisher's version: 10.1145/2699715

Permalink to record: http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-13-28342

A

TADSim: Discrete Event-based Performance Prediction for Temperature Accelerated Dynamics

Susan M. Mniszewski, Los Alamos National Laboratory
Christoph Junghans, Los Alamos National Laboratory
Arthur F. Voter, Los Alamos National Laboratory
Danny Perez, Los Alamos National Laboratory
Stephan J. Eidenbenz, Los Alamos National Laboratory

Next-generation high-performance computing will require more scalable and flexible performance prediction tools to evaluate software-hardware co-design choices relevant to scientific applications and hardware architectures. We present a new class of tools called application simulators, parameterized fast-running proxies of large-scale scientific applications using parallel discrete event simulation (PDES). Parameterized choices for the algorithmic method and hardware options provide a rich space for design exploration and allow us to quickly find well-performing software-hardware combinations. We demonstrate our approach with a TADSim simulator that models the Temperature Accelerated Dynamics (TAD) method, an algorithmically complex and parameter-rich member of the Accelerated Molecular Dynamics (AMD) family of molecular dynamics methods. The essence of the TAD application is captured without the computational expense and resource usage of the full code. We accomplish this by identifying the time intensive elements, quantifying algorithm steps in terms of those elements, abstracting them out, and replacing them by the passage of time. We use TADSim to quickly characterize the runtime performance and algorithmic behavior for the otherwise long-running simulation code. We extend TADSim to model algorithm extensions, such as speculative spawning of the compute-bound stages, and predict performance improvements without having to implement such a method. Validation against the actual TAD code shows close agreement for the evolution of an example physical system, a silver surface. Focused parameter scans have allowed us to study algorithm parameter choices over far more scenarios than would be possible with the actual simulation. This has led to interesting performance-related insights and suggested extensions.

Categories and Subject Descriptors: I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms: Discrete-event simulation, Distributed simulation, Performance

Additional Key Words and Phrases: Accelerated molecular dynamics, Temperature accelerated dynamics

ACM Reference Format:
S. M. Mniszewski, C. Junghans, A. F. Voter, D. Perez, and S. J. Eidenbenz, 2014. TADSim: Discrete Event-based Performance Prediction for Temperature Accelerated Dynamics. *ACM Trans. Model. Comput. Simul.* V, N, Article A (January YYYY), 25 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

This work is supported by the Los Alamos National Laboratory, under Laboratory Directed Research and Development.
Author's addresses: S. M. Mniszewski and C. Junghans and S. J. Eidenbenz, Computer, Computational & Statistical Sciences Division, Los Alamos National Laboratory; A. F. Voter and D. Perez, Theoretical Division, Los Alamos National Laboratory.

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

New computer architectures, software stacks, and algorithms will be necessary to further advance scientific research. These changes make challenges related to power consumption, cost of data movement, limited memory, data locality, extreme concurrency, and resilience increasingly pressing, especially as we progress towards exascale. Current architectural paths to exascale include multicore high-end CPUs such as Intel's X86, manycore/embedded simpler low power cores from embedded systems, and GPU/accelerated highly specialized processors from the graphics/gaming market space such as NVIDIA's Tesla and Intel's Xeon Phi Many Integrated Core (MIC). The High Performance Computing (HPC) field has become an adopter of these technologies. The HPC community, driven — among others — by the national security needs of the U. S. Department of Energy's (DOE) national laboratories, is involved in "computational co-design" efforts using their traditional software portfolio of physics applications to influence and be influenced by next-generation architectures [U. S. Department of Energy Office of Science 2012; Advanced Simulation and Computing Program 2012]. Computational co-design refers to the close coupling of hardware and software design. Co-design is a very well established set of methodologies in embedded systems [Wolf 1994]; extending these concepts to high-performance computing is a DOE science strategy [U. S. Department of Energy Office of Science 2014].

A view of co-design is that of a performance-guided search and selection from a design space of software and hardware options [Eidenbenz et al. 2012]. For each set of hardware-software choices, we predict the performance using scalable distributed discrete event simulation. The performance measures of interest are runtime metrics such as wall clock time (WCT), speedup, or energy usage. Based on these results, an optimization step — using either a scan of the design spaces or a guided search using optimization techniques such as genetic algorithms, tabu search, or simplex optimization methods — identifies the optimal instance in the search space. Parameterizations define the possible software and hardware options and each can be represented at a different level of fidelity. For example, when focusing on performance prediction of different algorithmic variants of the software implementation of a method, a coarse model may be sufficient at the hardware level.

The central role of performance prediction via simulation in our co-design approach is the main topic of this work, and here we focus primarily on exploring the software design space. We introduce the concept of a parameterized application simulator. The key stages of an application are modeled as discrete events, while abstracting time-intensive parts or kernels of the application. The logic of an application or pseudo-code is simulated, including loops, control flow, and termination conditions, similar to a state-machine (see Fig. 1). Designing an application simulator requires: identifying the time-intensive elements, quantifying the different algorithm steps in terms of those elements, abstracting them out, and replacing them by the passage of time. A low fidelity hardware architecture model is defined, considering parameters such as processor speed, core counts available, communications time, and thread overhead. Instrumentation is available for the collection of performance metrics. The software and hardware parameters specified in the simulator define the hardware and software design spaces that we explore. Application simulators allow for fast exploration of application design spaces, testing of new algorithmic extensions before actual implementation, and even testing of proposed hardware architectures that do not yet exist.

Here, we use an accelerated molecular dynamics (AMD) method, namely temperature accelerated dynamics (TAD), to illustrate this application simulation concept.

The development of this co-design approach was guided by this algorithmically complex and parameter-rich method.

2. THE APPLICATION SIMULATOR CONCEPT

The application simulator approach is most useful for performance analysis of applications that do not have predictable progression prior to runtime. Indeed, performance prediction of typical bulk synchronous parallel (BSP) approaches in most current-day physics simulation codes can often be accomplished by analytical methods. We view PDES as the most appropriate solution for modeling complex applications at the algorithmic level, when cycle-accurate results are not required. As we move towards next-generation HPC, asynchronous programming models are becoming more common, requiring alternative approaches. The use of PDES for performance prediction is then an attractive choice. This approach could be especially powerful in cases where the runtimes of certain sections of the codes are not constant, but can vary significantly from one case to the next (e.g., if a non-linear problem needs to be solved, or when the stop time of some procedure is a random variable). Assessing the performance of various implementation and parallelization strategies would be best approached by direct simulation, as analytical methods would rapidly become tedious.

An application simulator is ideally suited to represent an algorithm that spends most of its runtime performing some atomic unit of work. This can be a frequently used computationally intensive calculation, a data access/movement pattern, or a communication sequence. The algorithm can then be represented as a state diagram of a repeatable loop sequence of stages with a stopping condition, where each stage's runtime is specified in atomic units of work. This allows us to abstract out time-consuming computations, yielding a fast-running DES proxy. Stochastic decision can be included to drive branching, stopping conditions, asynchronous communication and/or spawning of subtasks. Software, hardware, and algorithm parameters are then defined. The parameters reflect the value of tunable constants or allow for different algorithmic choices. These parameters can be very specific or general based on the level of modeling abstraction desired. Instrumentation is added to collect performance, logic-based, and resource usage metrics. Due to the absence of executing work units, pre-computed information may be necessary to drive the simulation. Details of the dynamics can be obtained by mining the event progression of a simulation carried out with a standard version of the program.

Building an application simulator is currently a manual effort as automated tools are not yet available. Knowledge elicitation from domain experts is required.

3. RELATED WORK

The choice of parallel discrete event simulation (PDES) [Banks et al. 2008; Fujimoto 1990; Liu 2010] complements other performance prediction methods that include closed-form analytical models [Bauer et al. 2012; Spafford and Vetter 2012], semi-analytical models [Barker et al. 2006; Barker et al. 2009], simulation [Bagrodia et al. 1999; Rodrigues et al. 2011], emulation [Xia et al. 2004; Santhi et al. 2013], and hybrid simulation-emulation [Binkert et al. 2011; Calheiros et al. 2012; Zheng et al. 2004].

We note that the Scalable Simulation Toolkit (SST) [Rodrigues et al. 2011; Hendry and Rodriguez 2012] is a laudable example of PDES use in performance prediction. Unlike our application simulators, SST requires proxy application code as input and produces simulations that run slower than the actual application, which does not allow for exploring large parameter spaces and quick testing of new algorithmic ideas. The faster running SST/macro coarse-grain simulator runs a skeletonization of an application code for studying the effect of network parameters and topology on performance.

A skeleton preserves the control flow of communication code, but does not retain the domain algorithm behavior.

The simulation of software simulations is not new. The SIMSIM *meta-simulation* has been used for load-balancing of parallel and distributed simulation systems [Ewald et al. 2006] to understand behavior and performance characteristics. Application simulators are used to explore and understand the behavior of systems over many scenarios. A methodology was developed to predict the performance of a distributed simulation of an HLA middleware system prior to implementation [Gianni et al. 2010]. Likewise, the performance of algorithmic extensions can be evaluated prior to implementation using application simulation. The SONSim *second-order simulation* was used to predict performance for computer network simulations [Andelfinger and Hartenstein 2013]. Parameters related to different topologies (such as interconnected subnetworks, peer-to-peer networks, and wireless networks) and hardware choices (ex. Ethernet and Infiniband interconnects) were used to assess whether the simulation would benefit from parallelization. Software and hardware model parameter choices were available to explore the design space for application simulators and the abstract hardware model allowed for reasonably accurate performance predictions.

In cases where analytical models are too cumbersome, another valuable alternative is experimental algorithmics [McGeoch 2012]. The focus is on empirical methods to aid in building better algorithms and understanding performance given a set of conditions and assumptions [McGeoch 2007]. As an example, this approach has been used to understand cache performance for sorting programs and produce more cache-oblivious ones. Similarly, an application simulator of a physics code can be used to understand the performance under many parameterized scenarios not previously explored.

Simulation was shown to be a viable approach for performance prediction of stochastic algorithms using a machine-learning approach in [Jeschke et al. 2011]. Parametrized components or sub-algorithms serving as elements of a design space are composed in different combinations to represent the domain code of interest. The use of small benchmark models was shown to provide valuable insight into the algorithm performance.

Another co-design approach to design space exploration is driven by the notion of a mini-application or mini-app from the Mantevo project (mantevo.org). Here a mini-app is a self-contained proxy for a real scientific code that contains key performance aspects of this type of application. They are written to be amenable to re-factoring or change but representative enough to be useful in the scientific problem domain. These open source proxies are available to computer hardware vendors and software stack developers to study and improve application performance. They operate stand-alone or in simulated environments. In contrast, an application simulator is a virtualization of a scientific code in the form of an event-based simulator. The hardware environment is represented only at the level that is necessary to explore a given domain codebase. Some design choices can be expressed as parameterized options. Both approaches require re-factoring or rewrites for exploring different programming models and significant algorithm changes.

4. APPLICATION TO MOLECULAR DYNAMICS

As a first application of this methodology, we target atomistic simulation methods in the Molecular Dynamics (MD) family. MD is a computer simulation technique for modeling the physical evolution of interacting atoms by numerically solving their equation of motion. In order to carry out the integration, forces acting on atoms are derived from a potential, a so-called molecular mechanics force field. These calculations, commonly called force calls, are the computational core and dominant cycle-burner in any MD code. MD is widely used [Rapaport 2004] in material science [Steinhauser and Hi-

ermaier 2009], chemical physics [Clark 1985], and the modeling of biomolecules [McCammon and Harvey 1988].

Conventional molecular dynamics allows one to access time scales on the order of hundreds of nanoseconds to microseconds. Efforts to explore longer time scales have led to the development of Accelerated Molecular Dynamics (AMD) methods that provide access to time scales on the order of milliseconds to seconds or more. For many material systems, the dynamical evolution on these longer time scales is characterized by infrequent events, in which the system makes occasional transitions from one state to another (i.e., long periods of uneventful vibrations are punctuated by rare topology changes). The AMD methods exploit this infrequency characteristic to reach longer times.

Three such AMD generalizations have been proposed [Voter et al. 2002; Perez et al. 2009]. These techniques include simulating at a higher temperature to speed up transitions, parallelizing time through running multiple replicas in parallel, or modifying the interatomic potential in a controlled way. In all these cases, one has to then map the results (in a statistical sense) into the original, unbiased, conditions. AMD has been used to investigate a wide range of important phenomena occurring at the atomic scale, such as the velocity dependence of friction during friction force microscopy experiments [Li et al. 2011] and radiation damage annealing processes relevant for nuclear structural materials [Bai et al. 2010], to name only a few. The temperature-raising method, Temperature Accelerated Dynamics (TAD), which is a complex and algorithmically interesting AMD method, is the subject of the present work.

In our TAD application simulator, TADSim, we exploit the fact that the force call is by far the most computationally expensive part of any (A)MD method, accounting for at least 90% of all cycles. The runtime of a force call depends on the interatomic potential function (the gradient of which is proportional to the forces acting on each atom), the number of atoms, and the number of cores available. The force call is considered to be the atomistic time unit, abstracted out, and represented by the passage of simulation time, enabling fast simulations while allowing for reliable predictions of the application's run time. As shown in Fig. 1, the TAD algorithm is then expressed as a sequence of stages, each represented by an event, and whose cost is measured in required number of force calls. We focus at a detailed level of the algorithm using specific parameters, while the hardware is viewed at a higher level and described using more general parameters.

5. TEMPERATURE ACCELERATED DYNAMICS (TAD) METHOD

In this section we will introduce the algorithm and discuss the parameters used for performance prediction. For a detailed discussion of TAD, we refer the reader to [Sorensen and Voter 2000]. TAD is an algorithm for reaching long time scales in molecular dynamics (MD) simulations. With few exceptions, direct MD is limited to a maximum simulation time on the order of one microsecond, due to the requirement that the integration time steps are short enough ($\sim 10^{-15}$ s) to resolve the atomic vibrations ($\sim 10^{-13}$ s). For most materials, the dynamical evolution on these longer time scales is characterized by infrequent events, in which the system makes occasional transitions from one state to another. An individual state is a 3N-dimensional potential basin (where N is the number of moving atoms), shaped parabolically near the minimum. An example of such an event is the jump of a vacancy in a solid or an atom on a surface, and much more complex events, sometimes involving many atoms, can occur as well. The *accelerated molecular dynamics* methods, of which TAD is one, exploit this infrequency characteristic to reach longer times. We chose TAD for this initial study because its rich set of parameters make performance prediction at different settings difficult.

In a direct MD simulation, once an interatomic potential and appropriate boundary conditions have been specified, one simply integrates the classical equations of motion, perhaps modified to take into account coupling to a thermal bath at a desired temperature T. The MD trajectory, within its microsecond limitation, will then make these occasional transitions from state to state automatically. The system vibrates in this basin many times until a fluctuation causing a large excursion takes the trajectory over a ridge-top to an adjacent basin. There is a saddle point associated with this transition (the point on the ridge top with zero gradient of the potential energy and one negatively curved direction corresponding to the reaction path), and there is typically a large number of adjacent basins to which the transition can take place.

The key to reaching longer time scales for this kind of system is to pick one of the adjacent states, and an associated time, for the next transition. The probability for transitioning to a particular adjacent state is proportional to the *rate constant* for escape to that state. The rate constant to each of these states can be accurately approximated using transition state theory [Marcelin 1915], and if all the adjacent states are known, one escape path can be chosen at random, weighted by its escape rate. Because the first-passage time is exponentially distributed for such rare events,

$$p(t) = k\mathrm{e}^{-kt}, \tag{1}$$

where the exponent k is the sum of the rates over all escape paths out of the state, an appropriate time for the escape can be generated using

$$t_{\mathrm{Random}} = (1/k)\ln(1/r), \tag{2}$$

where r is a random number distributed uniformly on (0,1]. This stochastic procedure, known as kinetic Monte Carlo (KMC) [Bortz et al. 1975; Voter 2007], can be used to advance the system from state to state. The list of adjacent states to which the next transition might occur can be generated either by intuition, or by a procedure known as adaptive KMC, in which most or all possible saddle points are sought through randomly initiated searches [Henkelman and Jónsson 2001].

Here, we focus instead on the AMD approach: we let the trajectory itself find the transition event, as it would in direct MD if we waited long enough, but we modify the dynamics such that the trajectory picks this escape more quickly. The advantage is that only one escape path (or, in the case of TAD, a few escape paths) must be found, releasing us from the burden of trying to find all the escape paths. The challenge, though, is to design the modified dynamics in such a way that the relative probabilities of the different escape paths are preserved as accurately as possible. In the TAD method, this acceleration is achieved by raising the temperature of the system, while correcting for the tendency for transitions to occur in a different order at high temperature. In the following, we briefly describe TAD, and we define and explain the various TAD parameters we will vary in the TADSim simulations we present below.

In TAD, we advance the system from state to state at a temperature T_{Low} using information from simulations at a higher temperature T_{High}. Temperature control during these simulations is provided by a Langevin thermostat, e.g., by using the Langevin-Verlet integrator in [Allen and Tildesley 1989], which utilizes a sequence of random numbers. Starting from some configuration point (positions of all the atoms) R_{Start} in the initial state (state A) of the system, we first thermalize the system by evolving the trajectory for a time t_{Therm}, so that it loses its memory of the initial condition R_{Start} (block 1 in Fig. 1). At the end of each thermalization stage (and perhaps multiple times during the thermalization time), we interrupt the trajectory and perform a transition check (block 2 in Fig. 1), to verify that the trajectory is still in state A. If it fails this check (i.e., if it is found to be in a different basin), the trajectory is placed again at

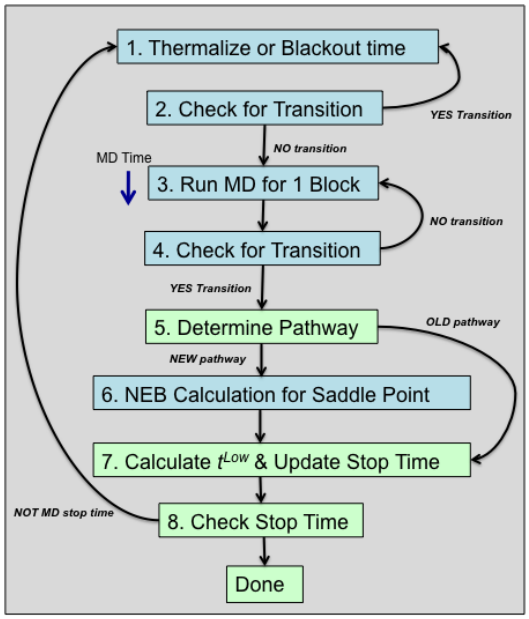

Fig. 1. Serial TAD algorithm as an execution flow of discrete events in TADSim. Blue blocks require force calls and green blocks do not.

some position inside state A, and the thermalization procedure is started from the beginning. We discuss below how transition checks are performed.

Assuming the thermalization stage has succeeded, so that we have prepared a properly thermalized trajectory in state A, the trajectory is continued, and official MD time t^{High} begins accumulating (block 3 in Fig. 1). At regular intervals, t_{Block}, the trajectory is interrupted to check for a transition (block 4 in Fig. 1), and this integration/detection cycle is repeated until a transition to some state other than A (e.g., state j) is detected.

Transition checks, which play an important role in TAD, are achieved by using the forces on the atoms to minimize the energy with respect to the geometry R (e.g., with a steepest-descent or conjugate gradient algorithm), and comparing it to R_{Min}, the known minimizer for the current state. If no transition has occurred, it typically takes a few to a few tens of force calls to conclude that the geometry is converging to R_{Min}, while if the system has made a transition to a new basin, much tighter convergence must be achieved before a transition can be declared, so a larger number of force calls is required. The determination of whether the transition to state j has been seen before takes place in block 5 of Fig. 1.

The time of this first transition to state j, t_j^{High} is taken to be a random time distributed uniformly on the interval of the most recent block of MD, i.e., $t_j^{\mathrm{High}} = t^{\mathrm{High}} - rt_{\mathrm{Block}}$, where r is a random number distributed uniformly on [0,1]. If this is a transition that has not been seen before, we then initiate a search to find the saddle point that connects state A with state j using the nudged elastic band (NEB) method [Jónsson et al. 1998] (block 6 in Fig. 1). In brief, the NEB approach relaxes a chain of configurations that connect the initial to the final state, in which each pair of adjacent configurations is connected by a $3N$-dimensional spring. At convergence, the highest-energy configuration in this chain gives a good approximation to the saddle-point energy.

A key concept in the TAD method is that each transition event observed in the trajectory at high temperature can be mapped onto a transition event in a corresponding hypothetical trajectory at the low temperature. At T_{Low} the time associated with the transition will be longer, and the order in which the transitions occur will in general not be the same as at T_{High}. However, within the harmonic approximation to transition state theory (HTST) [Vineyard 1957], the virtual transitions at T_{Low} generated in this way (the escape events and associated times) are in fact indistinguishable from those that would be generated by a long trajectory integrated directly at temperature T_{Low}.

In HTST, the escape rate to an adjacent state j is given by

$$k_j = \nu_{0j} \mathbf{exp}(-E_j/\mathrm{k_B}T), \tag{3}$$

where ν_{0j} is a temperature-independent pre-exponential factor, $\mathrm{k_B}$ is the Boltzmann constant, and E_j is the barrier height, $E_{\mathrm{Saddle},j} - E(R_{\mathrm{Min}})$. The pre-exponential factor can be computed from the normal-mode vibrational frequencies at the minimum and at the saddle point, although in the TAD method we don't need to do this. Note that in this approximation the only temperature dependence in the rate comes from the barrier height E_j. Thus, once we have found the saddle point associated with the event, we can compute the ratio of the rate constant at T_{High} to the rate constant at T_{Low}, and similarly we can compute the appropriately sampled *time* at T_{Low} using

$$t_j^{\mathrm{Low}} = t_j^{\mathrm{High}}(k_j^{\mathrm{High}}/k_j^{\mathrm{Low}}) = t_j^{\mathrm{High}} \mathbf{exp}(E_j(1/\mathrm{k_B}T_{\mathrm{Low}} - 1/\mathrm{k_B}T_{\mathrm{High}})). \tag{4}$$

This gives us a point on our low-temperature time line at position t_j^{Low} (block 7 in Fig. 1). We now repeat this procedure, beginning with a fresh thermalization in state A, and continuing to accumulate t_{High}. Each time we detect a transition, we find the saddle point and use Eq. 4 to place a point on the low-T timeline.

If we continue this high-temperature trajectory long enough, we will observe one or more events for every possible escape path out of state A, and each of these events will have an associated time at T_{Low} for this particular realization of the dynamics. We can identify the shortest-time event at T_{Low} as the transition that would have occurred first at T_{Low} (at a time $t_{\text{Low-Shortest}}$), and we can move the system to the state associated with this event.

The second key concept in TAD, the one that allows us to obtain a computational speedup (or *boost*) relative to direct MD, is that we can define a time, t_{Stop}, at which it is safe to terminate the high-temperature trajectory, knowing that with a desired confidence we have already observed the first low-temperature event. To define this stop time, we make the additional approximation that there is a lower bound ν_{Min} on the pre-exponential factors in the system, such that $\nu_{0j} \geq \nu_{\mathrm{Min}}$ for all reaction pathways out of A or any other state the system may visit. We also introduce an uncertainty parameter δ, which defines the confidence level $1 - \delta$ for our assertion that we have found the first event.

We can illustrate the TAD procedure, and derive the stop time, using an Arrhenius-like graphical representation. In a standard Arrhenius graph, one plots the logarithm of the rate against the inverse temperature, so that a reaction rate given by Eq. 3 corresponds to a downward sloping straight line with slope proportional to $-E_j/\mathrm{k_B}$ and intercept $\ln(\nu_{0j})$. We make an analogous plot, replacing the logarithm of the rate with the logarithm of inverse time. On such a plot, as shown in Fig. 2, the time progress of the trajectory at T_{High} corresponds to nonlinear downward motion on the vertical line at $1/T_{\mathrm{High}}$. For each escape event j at T_{High}, the time remapping given by Eq. 4 corresponds to extrapolation along a downward-sloping line with slope $-E_j/\mathrm{k_B}$ to find its intersection with the low-temperature timeline at $1/T_{\mathrm{Low}}$. It is easy to see that a high-barrier event extrapolates to a longer time at T_{Low} than a low-barrier event, and

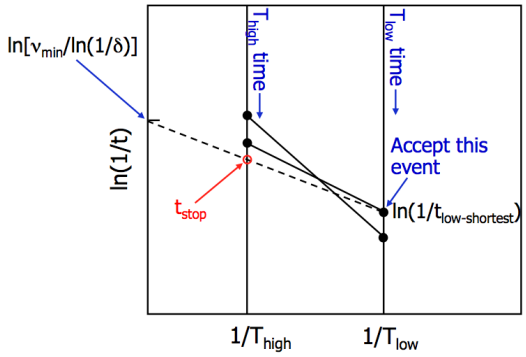

Fig. 2. Pictorial representation of the TAD method. Time moves down the vertical time lines shown. Each attempted transition at the high temperature is extrapolated to the low temperature along a line with a slope given by the negative of the activation energy (see Eq. 4). The MD simulation can be stopped, and the shortest-time event at low temperature can be accepted, when the high-temperature time reaches the intersection with the stop line (dashed), which connects the confidence-modified minimum prefactor on the y axis with the shortest-time event found so far on the low-temperature time line.

this gives rise to the possibility that the events occur in a different order at T_{High}. This is the main characteristic the TAD procedure must correct for: the first event at high temperature is typically not the first event at low temperature.

For any reaction pathway with a rate k_j, the probability distribution for the first event will be given by an exponential distribution, as in Eq. 1. The probability that we will see an escape along this pathway at least once before some time t' can be found by integrating Eq. 1 from zero to t', which gives the relationship

$$\mathrm{Prob}(\text{First event time} \leq t') = 1 - \exp(-k_j t'). \tag{5}$$

If we set this probability to $1-\delta$, and consider a rate with the simple temperature dependence given by Eq. 3, we obtain an expression for the time required to be $(1-\delta)$-confident of seeing the event,

$$t_{\mathrm{Conf}} = [\nu_{0j}/\ln(1/\delta)]\exp(-E_j/\mathrm{k_B}T), \tag{6}$$

On our Arrhenius-like plot, t_{Conf} corresponds to a line with slope $-E_j/\mathrm{k_B}$ and intercept $\nu_{0j}/\ln(1/\delta)$. As we move down the vertical time line at a given temperature, when we intersect this line, we have a confidence $1-\delta$ that a reactive event along this pathway will have occurred at least once.

We can now define the stop time, t_{Stop} (block 7 in Fig. 1). Once we have run the high-temperature trajectory long enough to see at least one event, we construct a "stop line", which connects the time of the current shortest-time event on the low-temperature timeline with a point on the y-axis corresponding to the *confidence-modified minimum pre-exponential factor*,

$$\nu^*_{\mathrm{Min}} = \nu_{\mathrm{Min}}/\ln(1/\delta), \tag{7}$$

noting that each time a new event is detected, the stop-line definition may change. The stop time is then the intersection of the high-temperature time line with this stop line. When t_{High} reaches t_{Stop} (block 8 in Fig. 1), the probability that any future event at T_{High} would create a shorter-time event at T_{Low} is lower than δ, so we can say with confidence $1-\delta$ that it is safe to accept the event at $t_{\mathrm{Low\text{-}Shortest}}$ and move the system to the new state.

This can be understood by considering the scenario in which we proceed with the MD trajectory infinitesimally beyond t_{Stop} and encounter an event X along a new pathway

with barrier height E_X. If E_X is greater than E_{Stop} (where $E_{\text{Stop}}/\text{k}_\text{B}$ is the the negative of the slope of the stop line), then the extrapolated time for this new event will be greater than $t_{\text{Low-Shortest}}$ and there is no problem – event X would not replace the accepted event. However, if E_X is less than E_{Stop}, then the extrapolated time would be shorter than $t_{\text{Low-Shortest}}$. However, by construction, the stop line we have just crossed is the $(1-\delta)$-confidence line for the reaction pathway corresponding to the worst-case possibility: $E_X = E_{\text{Stop}}$ and a $\nu_X = \nu_{\text{Min}}$. A reaction pathway with precisely this barrier and prefactor probably does not exist in the system, but we can say with $1-\delta$ confidence that if it did we would have already seen an event for this pathway. If a slightly different pathway exists, e.g., with a higher prefactor, or a lower barrier, then the probability is even greater than $1-\delta$ that we would have already observed this pathway. Thus, overall, we can say that the probability is smaller than δ that we are accepting the wrong event by terminating the trajectory at t_{Stop}.

Assuming that δ and ν_{Min} are chosen wisely, the main source of inaccuracy in the TAD method arises from *anharmonicity*, the discrepancy between the actual transition rate for a given pathway and the HTST predicted value for that rate from Eq. 3, with its simple temperature dependence. This discrepancy (positive or negative), which is negligible at low T and typically increases in magnitude with T, causes an error in the predicted times at T_{Low} when Eq. 4 is used to convert to the low temperature time for the event.

The TAD method has been implemented in multiple codes, some of which are widely used in the computational physics community [Plimpton 1995].

6. TADSIM - MODELING TAD AS A DISCRETE EVENT SIMULATION

TADSim – our parameterized application simulator – models the functionality of the TAD method as described in the previous section through DES of the individual stages of TAD, a flow-chart of which is shown in Fig. 1. Each stage becomes a discrete event in TADSim, which upon being processed will create future events. The design of TADSim relies on abstracting out the computationally intense parts and replacing them by the passage of time, while keeping a realistic execution flow in place. The force call is the most computationally intensive part of TAD and any other MD method. In fact, compute cycle consumption is dominated by force calls in each stage of TAD, so that the cost of the different stages of TAD is simply proportional to the number of force calls that they consume. This is the key idea behind TADSim.

In the following, we present TADSIM details by explaining the parameters available, the execution flow of the TAD simulation, the performance prediction metrics, and the underlying simulation engine.

6.1. Design Space Parameters

Parameters that we can vary allow us to explore the hardware and software design spaces. We divide the input parameters into four categories: (i) *Physical System Parameters*, (ii) *Method Parameters*, (iii) *Architecture Parameters*, and (iv) *Simulation Engine Parameters*. Table I lists all the available parameters. A description of each category and its associated parameters follow.

Physical System Parameters describe the material system that is the target of the TAD simulation. `NAtoms` is the number of atoms simulated. $T_{\texttt{Low}}$ is the low temperature at which the physical system is being simulated. `FcClockCyclesPerAtom` is the number of clock cycles per atom for each force call. The total number of clock cycles used in a single force call is then `FcClockCyclesPerAtom` $\times$ `NAtoms`. The actual value of this parameter depends on the complexity of the potential.

A rate catalog of pathway information is precomputed using TAD machinery. This rate catalog gives the characteristics of the various possible transitions that allow the

Table I. TADSim Simulation Parameters

Name	Description
Physical System Parameters	
NAtoms	# of atoms in system
T_{Low}	Low temperature (K)
FcClockCyclesPerAtom	# of clock cycles per atom for force call computation
BarrierHeight	Barrier height for each pathway in rate catalog (eV)
Prefactor	Prefactor for each pathway in rate catalog
NEBCalls	# of NEB force calls for each pathway in rate catalog
AnharmonicityCorrection	Correction applied to pathway rate for anharmonicity
AnharmonicityMinTemp	Minimum temperature (K) for anharmonicity adjustment
FcYesTransition	# of force calls when a transition is detected
FcNoTransition	# of force calls when a transition is NOT detected
FcTransitionWidthFraction	Fractional width of normal distribution for transition force calls
Method Parameters	
MDTimestep	MD Timestep (s)
MinPrefactor	Minimum state/pathway prefactor, ν_{Min}
Delta	Uncertainty value, δ
T_{High}	High temperature (K)
FcBlock	# of force calls per MD block
FcThermalize	# of force calls per thermalize
FcThermalizeCheck	Frequency of transition checks during thermalize (force calls)
FcCoreCount	# of cores used for MD computation
TransCheckSpawnCoreCount	# of cores to use when spawning a transition check
NEBBeadSpawnCoreCount	# of cores to use per bead when spawning a NEB
NEBBeadCount	# of beads for NEB saddle point calculation
Architecture Parameters	
TotalCoreCount	Total cores available
CoreClockSpeed	Clock speed of cores (GHz)
CommDelay	Communications delay during NEB (μs)
SpawnDelay	Delay before and after spawning (μs)
Simulation Engine Parameters	
TAD_COUNT	# of TAD trials to run
SEED	Random seed
TIME_FACTOR	Smallest time unit in DES system (fraction of μs)
END_TIME	End of simulation (based on TIME_UNIT)

system to exit from the initial state. BarrierHeight is the energy barrier (the difference in energy between the saddle point and minimum) and Prefactor is the temperature-independent pre-exponential factor. NEBCalls is the number of force calls required to find the saddle point using the NEB method. AnharmonicityCorrection is a parameter used to determine the anharmonicity in the pathway rate when T_{High} is higher than the AnharmonicityMinTemp. The anharmonicity correction factor, which multiplies the rate, is given by $(1 + (T - \texttt{AnharmonicityMinTemp}) \times (\texttt{AnharmonicityCorrection} - 1)/\texttt{AnharmonicityMinTemp})$, where AnharmonicityCorrection is path dependent. The number of force calls required to complete a transition check is represented by a normal distribution, with mean FcYesTransition and FcNoTransition for the positive- and negative-resulting transition checks, respectively, and relative width FcTransitionWidthFraction. FcYesTransition is much larger because a complete minimization is required when a new state is discovered, while a negative result can be determined quickly. FcNoTransition also has a temperature dependent factor, given by the same expression as above for the rate anharmonicity, with AnharmonicityCorrection=6.0. These values and characteristics are based on the observed TAD behavior.

Method Parameters are software or algorithmic parameters that define the method variation (in our case a variation of TAD) that we are evaluating, and hence describe the software side of the co-design loop. This category contains the most parameters

that we will vary in parameter scans in later sections. MDTimeStep, MinPrefactor, Delta, and $T_{\mathtt{High}}$ have been defined in Section 5. FcThermalize is the number of force calls required to thermalize the system when the trajectory is replaced in the state after a transition. Transition checks are performed every FcBlock and FcThermalizeCheck force calls, during the MD and thermalization stages, respectively. The early detection of a transition during thermalization and/or running MD reduces the amount of wasted MD time, but frequent checks incur additional cost, unless the transition check is spawned off, as we discuss below.

FcCoreCount is the number of cores used for running a force computation in parallel. It allows us to test an extension to basic TAD, namely parallelizing a force call. For the modest system sizes that are typically of interest with TAD, the most common parallelization is a force decomposition strategy. The speedup of this approach is limited by synchronization and reduction costs, and hence saturates upon strong scaling. We model the speedup achieved by using FcCoreCount cores as $\sqrt{a \times \mathtt{FcCoreCount}^2/(a + \mathtt{FcCoreCount}^2)}/\sqrt{a/(a+1)}$, where a = 100. This function initially increases rapidly, but saturates at a speedup of 10 for large FcCoreCount.

NEBBeadCount is the number of beads used in the NEB method (i.e., the number of configurations in the chain). The NEB force calls are executed by each bead in parallel. Additional parallelism is achieved as each bead's force calls can be executed by a separate set of cores.

TransCheckSpawnCoreCount and NEBBeadSpawnCoreCount specify how many cores can be allocated to transition checks and NEB computations (per bead) respectively, when these computational stages are spawned as separate processes. We describe this in more detail in Section 8.

Architecture Parameters characterize the hardware platform. Our hardware model matches the level of abstraction that the software model provides. The architecture parameters describe an architectural model of a large compute cluster consisting of a number of cores (TotalCoreCount), each running at the same clock speed (CoreClockSpeed). The simulated WCT that a force call takes to execute in TADSim is the product of CoreClockSpeed, NAtoms, and FcClockCyclesPerAtom. Spawning or creating a new process incurs a delay (SpawnDelay). If processes on different cores communicate with each other, a communication delay is incurred (CommDelay). As we focus mainly on method parameters for TADSim, our hardware model is not very detailed by design.

Simulation Engine Parameters control the internals of the underlying simulation engine. These are largely self explanatory, and we note that the TIME_FACTOR is typically set to 1 μs.

6.2. Execution Flow

TADSim keeps track of multiple time scales simultaneously: simulated wall clock time (WCT) for the TAD runtime; MD time at high temperature, collected only when running MD; and MD time at low temperature, as updated after each NEB is calculated. Following the stages of TADSim in Fig. 1: Thermalization attempts are repeated until no transition is observed during FcThermalize. Then MD is run one block at a time (FcBlock MD steps), until a transition check indicates that a transition occurred. The high temperature MD time is backed up by a random fraction of a block to correct for the overestimation of the real transition time. The transition pathway is determined randomly using the pre-calculated pathway rates and the algorithm described below. If this pathway was not observed previously, the NEB calculation for the saddle point is performed, at a cost that depends on the nature of the transition. Then, a new putative MD stop time is calculated based on the new TAD low temperature MD time according

Table II. TADSim Performance Prediction Metrics

Name	Description
Algorithm Metrics	
# of MD force calls	Total force calls used when running MD
# of Transition Check force calls	Total force calls used for transition checks
# of Thermalize force calls	Total force calls used for running thermalize
# of NEB force calls	Total force calls used for NEBs
# of force calls	Total force calls
# of new pathways	Total unique pathways seen
# of repeat pathways	Total pathway replicates seen
TAD Shortest time	Final Low temperature MD time (s)
MD time	Total MD time (s)
MD WCT	Total WCT for MD (s) when running on 1 core
Raw Boost	TAD shortest time / MD stop time
Computational Boost	MD WCT / WCT
Computational Boost Error	Standard error for Computational Boost
Architecture Metrics	
Maximum Cores Used	Most cores used in parallel during simulation
# of Spawned Transition Checks	Total transition checks spawned
# of Spawned NEBs	Total NEBs spawned
Communication Cost	Total messages (used when spawning)
WCT	Total runtime (s)

to the TAD equations of Section 5. This becomes the official stop time if it is smaller than the previous value. These stages are executed iteratively until the MD stop time is reached.

The stochastic nature of TADSim lies in the transition checks: Transition checks during thermalization and running MD involve determining whether a transition occurred since the last check, if so, which of the possible transitions occurred, and how long the transition check lasted. The probability P that any transition would occur during the MD block is given by

$$P = 1 - \exp(-k_{\text{Total}} \times \texttt{MDBlockTime}), \tag{8}$$

where `MDBlockTime` = `FcBlock` $\times$ `MDTTimestep`. If P exceeds a random number drawn on [0,1], then a transition has occurred. Since the probability for each particular transition is proportional to its rate, the index of the transition that occurred can then be obtained by drawing another random number on [0,1] and identifying the interval of the partial cumulative sum of relative rates where it is located. Finally, the duration of a transition check is drawn from a normal distribution with specified mean (different for positive vs. negative outcomes) and width, as discussed above.

6.3. Performance Prediction Metrics

TADSim has been instrumented to collect the algorithm and architecture prediction metrics from each TAD trial as shown in Table II. Individual TAD metrics are reported as well as average metrics across all TADs at run completion.

The algorithm metrics provide a means of comparison with the actual TAD simulation runs. The Raw Boost provides the speedup of the TAD method based on MD time alone (i.e., the amount of high temperature MD time that was required divided by the low temperature TAD time that was achieved). This does not include the overhead from thermalization and NEBs, and lost time (one half of a block on average) when a transition is detected. This overhead is included in the Computational Boost, defined as the ratio of the WCT it would take to run the same amount of low temperature MD time on one core to the predicted TAD WCT. Force call counts are also provided per algorithm stage.

The architecture metrics provide information relevant to the computer architecture being studied, such as the maximum number of cores used in parallel, spawning costs in terms of counts and communication cost, and WCT. The WCT is the predicted runtime of the TAD simulation given the input configuration.

6.4. Simulation Framework

We have used the C++ SimX (formerly known as SimCore) PDES Framework [Kroc et al. 2007; Mniszewski 2010] for the implementation of TADSim. SimX is a library for building large-scale distributed-memory, discrete event simulations using the discrete event engine from the Parallel Real-time Immersive Modeling Environment (PRIME) [Modeling & Networking Systems Research Group, Florida International University 2012] for passing events, event queue maintenance, and synchronization. The important concepts and classes within SimX are Entity, Service, Info, and Profile. An Entity is a class that represents a simulation object, such as a Controller or TAD. A Service is a class that is used to implement the behavior of an Entity and operates like an event handler. Services are attached to Entities. An Info is a class that represents an event that can be scheduled and supplies additional data items and is processed by a Service. Infos are passed between Entities (more typically between Services) to trigger an action. A Profile is a way of providing runtime specification of default parameter settings for different types of Entities, Services, and Infos.

There are two kinds of simulation objects or Entities in TADSim, a TAD Entity and a Controller Entity. A TAD Entity performs the TAD algorithm as shown in Fig. 1. Multiple TAD Entities can run in parallel, each representing a separate trial, and no communication is required between them. The simulation can run in a distributed fashion with a Controller Entity on each CPU. The Controller Entity's job is to start the TAD Entities, coordinate transitions, collect prediction metric data from the TAD Entities, and report average prediction metrics when done. When running distributed, the Controller Entities operate as a hierarchy for passing on metric results.

7. VALIDATION

As a test of our approach, we compare the results of TADSim simulations with real TAD simulations on an actual atomistic system. For simplicity, we choose to investigate a silver adatom (an extra isolated atom) on a silver (100) surface (see Fig. 3). The system contains 301 atoms, of which 151 are free to move (the bottom layers are held fixed at their equilibrium position). The interatomic potential is of the embedded atom method (EAM) form [Daw and Baskes 1984; Voter 1994], using a parameterization for silver [Voter 1988]. By simulating the system using molecular dynamics at a high temperature (T = 1200 K) for 25 ns, we identified 1379 possible transitions. The barrier height and pre-exponential factor were determined for each pathway, as needed to compute the HTST rate at any temperature using Eq. 3. The NEB calculation for each pathway was deployed in the same way it would be in an actual TAD simulation, to obtain values for `NEBCalls`. In this system, the dominant transition pathway (i.e., the one with the lowest barrier of 0.492 electron volts (eV)) is a hopping mechanism that takes the adatom to a nearest-neighbor binding site. There are four of these, as there are four equivalent directions on the Ag(100) surface. At slightly higher energy (0.586 eV) there is an exchange pathway in which the adatom plunges into the surface and pushes a substrate atom up into the next-nearest neighbor position. All the remaining pathways discovered in the MD involve two or more atoms and have barriers in the range of 0.98 eV to 1.65 eV. Anharmonicity corrections were determined, for just the hop and exchange events, by comparing the observed high temperature rates to the HTST rates, giving `AnharmonicityCorrection` = 1.5 for the hop and `AnharmonicityCorrection` = 4.0 for the exchange (i.e., the MD hop rate is 1.5 times faster than the HTST hop rate at T

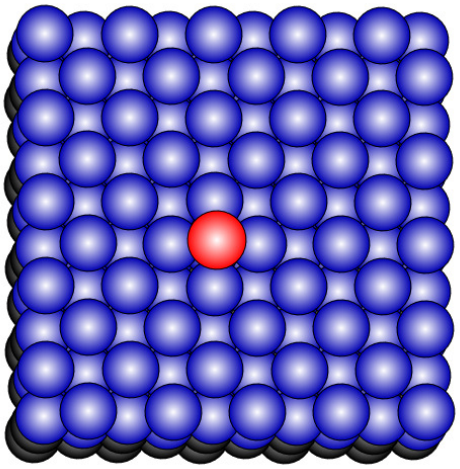

Fig. 3. Atomistic system used for the validation of TADSim, corresponding to an adatom on a silver (100) surface. The moving atoms are blue, the non-moving atoms are black and the adatom is shown in red. The adatom can hop to an adjacent binding site, or perform a two-atom exchange to a next-nearest neighbor site. A large number of higher-barrier processes are also available to this system.

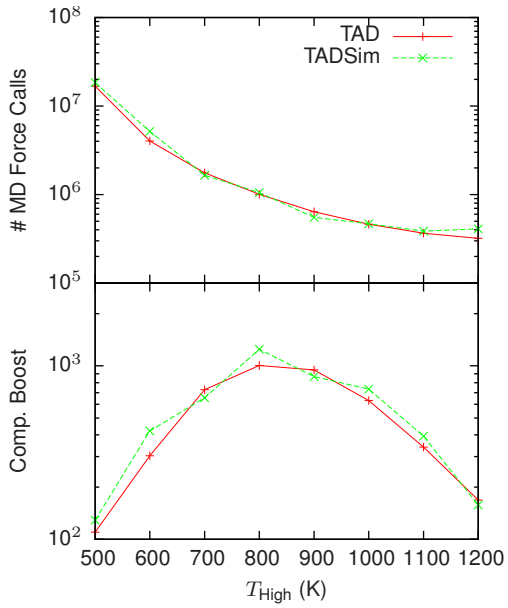

Fig. 4. Comparison of number of MD force calls and Computational Boost for TAD vs. TADSim.

= 1000 K, and the exchange rate is 4 times faster). The resulting catalog of rates was then used by TADSim.

TAD and TADSim were run using the parameter values in the second column of Table III. TAD currently runs on only 1 core; likewise, the total available cores in TADSim were limited to 1. 60–64 trials were run for high temperatures ranging from 500 K to 1200 K. The average number of force calls per algorithm stage and the Computational Boost were compared. TAD was run in serial on 2.67 GHz Intel(R) Xeon(R) CPU X5650 processors.

The results, shown in Fig. 4, demonstrate an excellent agreement between TAD and TADSim results. For example, the number of forces calls taken during MD stages are in excellent agreement over the whole range of high temperatures and properly account for the $\sim$ 100-fold change in this temperature range. We observe a similar agreement in terms of the overall Computational Boost, which is the main metric of interest in this study. Again, TADSim properly reproduced the temperature variation of the boost, including the location of the maximum, and the absolute values are in excellent agreement. We note that the agreement is not as good without the rate anharmonicity corrections.

Incidentally, these results illustrate the challenge in predicting the performance of TAD: while the number of force calls decreases monotonously with increasing high temperature (a consequence of the fact that the expected stop time decreases as T_{High} increases), the increasingly large overhead from the thermalization, transition checks, and NEBs counteract this gain, leading to an optimal performance at finite temperature. Minimizing the impact of this type of overhead on the run time of TAD is the key to improving the performance.

These results clearly demonstrate that TADSim reliably predicts performance trends and can be used as a good performance predictor in a systematic co-design

approach. This also validates the abstraction of the TAD algorithm that underpins TADSim.

8. EXPLORING NEW ALGORITHM EXTENSIONS

As previously noted, mitigating the impact of overhead is key to performance. In that respect, transition checks and NEBs are prime candidates for optimization, given their high cost (up to 500 and 20,000 force calls, respectively, for the validation example).

In usual conditions, most transition checks return with a negative result. It can therefore be beneficial to spawn off a transition check and start running the next MD block, speculating that the transition check will indeed return a negative result. This allows for the concurrent execution of the MD and transition checks. Once a transition is seen, the current MD block is aborted and TADSim rolls back to the point where the MD block that resulted in the positive transition check ended (which can be a few MD blocks in the past, depending on the duration of the check). This is known as a speculative programming model [Tapus and Hickey 2007]. It does require the use of extra cores and start/stop overhead time for the spawning, but can lead to a reduction in overall runtime.

Likewise, this same idea can be applied to NEBs. As pathways are seen for the first time, NEBs can be spawned and the next thermalization started, potentially resulting in multiple NEBs executing simultaneously. The stop times are updated as the results of the NEBs become available, and, as soon as it can be established that the MD stop time has already been reached by the ongoing MD, unfinished tasks are aborted and the simulation ends. Spawning NEBs can lead to a significant improvement in the run time.

Speculative spawning requires specifying the number of cores used for a spawned transition check (`TransCheckSpawnCoreCount`) and a spawned NEB (`NEBBeadSpawnCoreCount`). When spawning is attempted, TADSim insures that a sufficient number of cores are available to complete the request. When not enough cores are available, transition checks are performed serially. In the case of NEBs, the availability of cores is reassessed after a preset delay, until spawning is possible. A delay before and after spawning (`SpawnDelay`) represents the start-up and tear-down of lightweight vs. heavyweight threads.

9. RESULTS

TADSim can be used to predict TAD performance given a set of parameter choices as part of an optimization process or a parameter scan. In the following, the primary performance metric is the Computational Boost, which is the speed-up over standard MD on a single core, including overhead. In our co-design context, a set of parameter values defines a variation of TAD on the software side, as well as an architecture on the hardware side. While a standard parameter optimization procedure, such as a genetic algorithm, could be used for such a task, such a single-path optimization run would not characterize the response surface of the performance function over these hardware and software design spaces very well. In particular, it would not allow us to explicitly study tradeoffs between different parameters. However, if the goal is simply to identify optimal parameters without rationalizing the nature of the optimum, an optimization approach is preferable.

We execute parameter scans in a fully factorial fashion over a suitably discretized design space, leveraging access to supercomputing resources. The resulting high-dimensional description of the response surface, which is the dependence of performance on software and hardware parameters, enables us to discover the traces of performance phenomena. However, any factorial design inevitably results in a very significant CPU time consumption. This is addressed by initially running a coarse scan for

Table III. TADSim Parameter Scan Values

Name	*Validation*	*Coarse Scan Values*	*Detail Scan Values*
Physical System Parameters			
`NAtoms`	151 atoms		
$T_{\mathtt{Low}}$	300 K	200 K, 300 K	300 K
`FcClockCyclesPerAtom`	8000 cycles		
`BarrierHeight`	Precomputed per pathway in rate catalog		
`Prefactor`	Precomputed per pathway in rate catalog		
`FcNEB`	Precomputed per pathway in rate catalog		
`AnharmonicityCorrection`	Precomputed per pathway in rate catalog		
`AnharmonicityMinTemp`	500 K		
`FcYesTransition`	500 force calls		
`FcNoTransition`	10 force calls		
`FcTransitionWidthFraction`	0.25		
`FcThermalize`	500 force calls		
`NEBBeadCount`	21 beads		
Method Parameters			
`MDTTimestep`	4 fs		
`MinPrefactor`	10^{11} s^{-1}		
`Delta`	0.01		
$T_{\mathtt{High}}$	400 K - 1200 K in 100 K steps	400 K, 600 K, 800 K, 1000 K, 1200 K	800 K, 850 K, 900 K, 950 K, 1000 K, 1050 K, 1100 K, 1150 K, 1200 K
`FcBlock`	500 force calls	1, 2, 10, 50, 100, 400	1, 2, 5, 20, 100
`FcThermalizeCheck`	500 force calls	1, 2, 10, 50, 100, 400	1, 2, 5, 20, 100
`FcCoreCount`	1	1, 16, 64, 512, 4096 cores	64 cores
`TransCheckSpawnCoreCount`	0	0, 1, 16, 64, 256 cores	64 cores
`NEBBeadSpawnCoreCount`	0	0, 1, 16, 64, 256 cores	64 cores
Architecture Parameters			
`TotalCoreCount`	1	1, 64, 512, 4096, 32768, ∞ cores	∞ cores
`CoreClockSpeed`	2 GhZ		
`CommDelay`	1 μs		
`SpawnDelay`	1000 μs		

a discrete set of parameters, followed by a tightly meshed, fully factorial scan around the peak performance point. We further define focused scans on just a few parameters to precisely assess the nature of the performance phenomena.

The evaluation was performed on an HPC network using multiple nodes for simulation. The cluster is composed of nodes with 4 Quad-Core AMD Opteron 8354 Processors operating at 2.2 GHz interconnected using Infiniband 4X Dual Data Rate DDR network. Single core DES runs were distributed across multiple nodes using GNU Parallel (www.gnu.org/software/parallel/).

9.1. Parameter Scans to Explore the Performance Response Surface

Here we present results from fully factorial parameter scans of TADSim hardware and software parameters with the values as given in the third column of Table III. Only values that differ from the validation run are shown. The selection of parameters to be varied was chosen by our domain experts. Preliminary runs identified parameters that had little effect on performance, such as `CommDelay`. This indicates that our variations of TAD are not communication-bound. Use of ∞ for `TotalCoreCount` allowed us to explore extreme or unlimited situations. The small values for the number of force calls between transition checks during thermalization (`FcThermalizeCheck`) and running MD (`FcBlock`) represent similarly extreme situations, as they would be very inefficient in serial. Overall, our parameter values generated millions of scenarios. We performed 64 independent runs for each scenario using different random seeds to obtain averaged performance metrics. Partial results of the scan are presented in Fig. 5

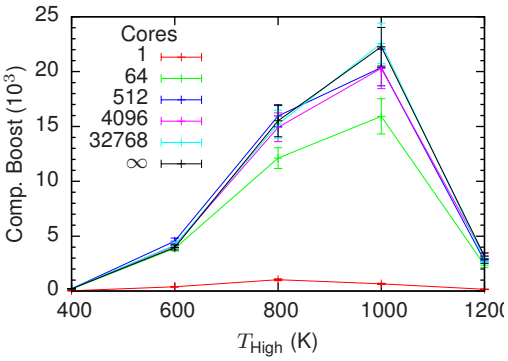

Fig. 5. Maximum Computational Boost as function of $T_{\mathtt{High}}$.

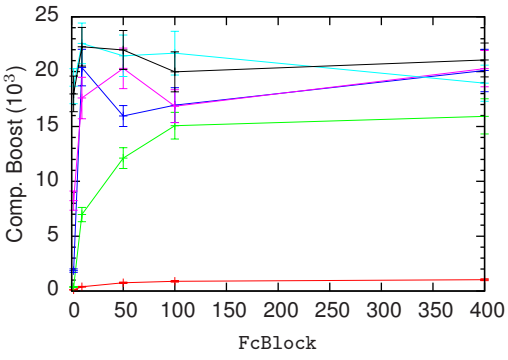

Fig. 6. Maximum Computational Boost as a function of FcBlock (Legends are the same as in Fig. 5.)

and Fig. 6. In these figures, we vary one parameter, $T_{\mathtt{High}}$ or FcBlock, and plot the corresponding maximum Computational Boost over the rest of the parameter space. Low temperature ($T_{\mathtt{Low}}$) is set to 300 K. Results are shown for different TotalCoreCount limits.

Because the results in Figs. 5 and 6 represent scenarios giving the maximum Computational Boost taken from data with statistical error bars, we require an additional procedural step to eliminate bias in the results. For a given parameter constraint (e.g., $T_{\mathtt{High}}$ and TotalCoreCount) in Fig. 5), the optimum scenario chosen from all possible combinations of the remaining parameters will have the largest Computational Boost not solely due to having the best parameter settings, but instead due to a combination of good parameter settings and a favorable (positive) statistical fluctuation. Consequently, this value will be biased in the positive direction. To eliminate this bias, after selecting the maximum-boost scenario, we recompute the Computational Boost for that scenario with a freshly-chosen random number seed, and we plot this value. Each result shown thus represents a lower-bound on the true value, with statistical error bars, where the lower-bound nature arises from two sources: 1) the procedure just described can lead to selection of a less-than-optimal grid point and 2) the parameter grid has finite resolution.

Fig. 5 shows the performance of TAD variations of different maximum core counts (TotalCoreCount) at different high temperatures ($T_{\mathtt{High}}$). Consistently, peak performance is seen at a $T_{\mathtt{High}}$ of 1000 K for each of the TotalCoreCount values. The corresponding plot for $T_{\mathtt{Low}}$ = 200 K (not shown) is qualitatively similar but has a less pronounced peak. Even with this coarse scan we can state that 64 cores are sufficient to reap a majority of the parallel performance possible. Values of TotalCoreCount beyond 64 provide an additional improvement of $\sim$40%. We note that the lower boost at 64 cores is limited by a lower number of CPUs available to do MD (FcCoreCount = 16, giving 8.5x speedup compared to a single core, out of a maximum possible speedup of 10x) and fewer CPUs for spawning transition checks (TransCheckSpawnCoreCount = 16).

Fig. 6 shows the dependence of the performance on FcBlock, the time interval between transition checks. We note again the benefit of allowing additional cores, especially for very frequent transition checks. Interestingly, we see that performance is similar for all values of of FcBlock for larger core counts, as all transition checks and NEBs are spawned. Lower core counts result in a mix of spawned and not-spawned transition checks, producing lower performance when FcBlock is small.

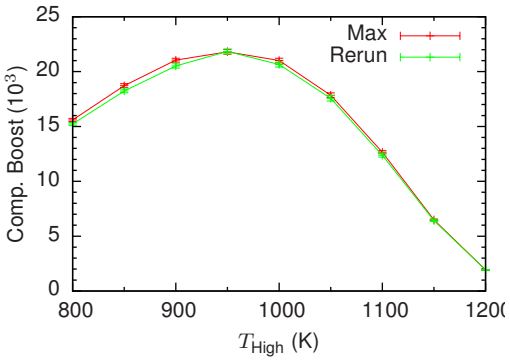

Fig. 7. Maximum Computational Boost vs. T_{High} in the detailed scan. Optimum performance is achieved at 950 K.

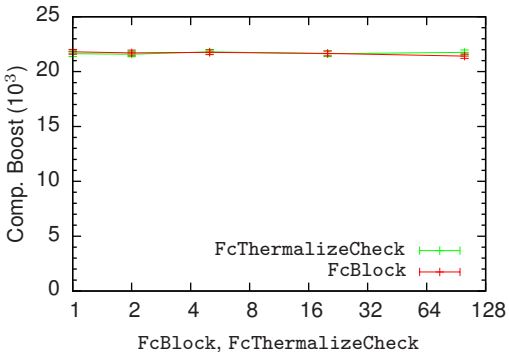

Fig. 8. Maximum Computational Boost vs. block size in the thermalization stage and the MD stage for the detailed scan.

The fully factorial scan provides (i) a strong indication of the location of the optimal performance area and (ii) uncovers performance behavior phenomena that merit further study. In the following subsections, we present the results of a fully factorial tightly meshed scan around the optimum parameter values suggested by the coarse scan and we describe a series of low parameter dimension studies to illuminate the performance behavior found in the coarse scan.

9.2. Detailed Scan Around the Optimum Parameter Values

For the detailed scan, we vary the parameters around the optimal values found in the coarse scan as described in the fourth column of Table III. We applied the same unbiasing procedure as above, so again these points represent lower bounds on the exact results, with statistical error bars, although now the bounding and the error bars are much tighter (e.g., see Fig. 7, which shows both the maximum points (red) as well as the points resampled with fresh random number seeds (green)). Fig. 7 shows the performance results with respect to T_{High} when T_{Low} = 300 K. We find that for our silver physical system, the optimum high temperature is 950 K. Fig. 8 shows the variation of the boost upon varying the time between transition checks during thermalization (`FcThermalize`) or MD (`FcBlock`). In this optimization with unlimited cores, any dependence on `FcBlock` virtually disappears over the range considered (`FcBlock`$\leq$100). This offers the user the opportunity to make more frequent transition checks to improve the physical time-scale resolution, which can be important to the accuracy of the final physical predictions in some cases. However, as discussed above (and also see below), greater total core counts are required for the small values of `FcBlock`.

9.3. Exploring Performance Phenomena: One-dimensional Parameter Scans

The fully-factorial coarse scan points to performance phenomena that warrant further study. We focus on a few scenarios in detail, varying only a few parameters, and we organize our findings along observations.

OBSERVATION 1. *The optimal high temperature T_{High} decreases with increasing T_{Low}.*

Our observation of how T_{Low} and T_{High} relate is at first glance surprising and appears counter-intuitive. More in-depth theoretical analysis has confirmed that finding to be accurate. We show in Fig. 9 how the optimum high temperature T_{High} increases with decreasing T_{Low}. The logarithmic y-axis scale is noteworthy, as it illustrates how much

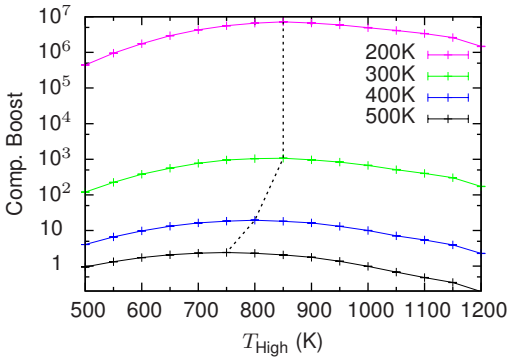

Fig. 9. Computational Boost as function of T_{High} for different T_{Low}. The dotted line passes through the optimum T_{High} for each T_{Low} curve.

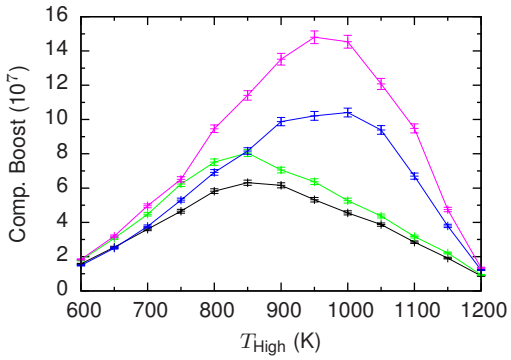

Fig. 10. Computational Boost for no spawning (black), spawning of transition checks (green), spawning of NEBs (blue), and spawning both (red), at T_{Low} = 200 K. Here 64 cores are used for the MD, and 64 cores as well for each spawned transition check and each NEB bead. (Legend is the same as in Fig. 11.)

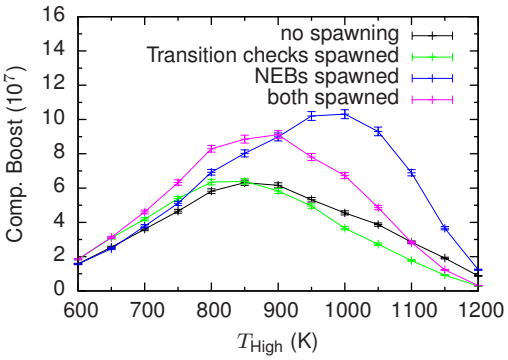

Fig. 11. Computational Boost for no spawning, spawning of transition checks, spawning of NEBs, and spawning both, at T_{Low} = 200 K. Here 64 cores are used for the MD, while only one core is used for each spawned transition check and each NEB bead.

more Computational Boost we gain for lower temperatures: more than three orders of magnitude between the low temperatures of 300 K and 200 K. This is expected given that the exponential decrease of transition rates with decreasing temperature tremendously increases the potential for acceleration. The results also show that the optimum $T_{\mathtt{High}}$ moves from 850 K at a $T_{\mathtt{Low}}$ of 200 K to 750 K at a $T_{\mathtt{Low}}$ of 500 K. This can be explained by the following: for a given system, a given setting for $T_{\mathtt{High}}$ and $T_{\mathtt{Low}}$ leads to a particular value for the average slope of the "stop line" (dashed line in Fig. 2), the line connecting shortest-time event at $T_{\mathtt{Low}}$ with $\nu_{\mathrm{Min}}*$ on the y axis. For typical values of ν_{Min} and δ, if $T_{\mathtt{low}}$ is increased, the average slope of this stop line will decrease in magnitude (become less negative), because the time of the shortest-time event decreases. This weaker slope means that if we consider lowering $T_{\mathtt{High}}$, which has the benefit of reducing the average number of NEBs required before the stop time is reached, we can do so with less of a penalty in terms of lost boost. Thus, if $T_{\mathtt{High}}$ was chosen optimally for the first value of $T_{\mathtt{Low}}$, this implies that for a higher $T_{\mathtt{Low}}$, the new optimum $T_{\mathtt{High}}$ is lower. Another consequence of this is that if $T_{\mathtt{Low}}$ is raised far enough,

the slope of the stop line can actually become positive, at which point the optimum setting for $T_{\texttt{High}}$ is $T_{\texttt{High}} = T_{\texttt{Low}}$, indicating that direct MD will be faster than TAD.

We note that the optimum high temperature in Fig. 9 is different (lower) than in our factorial scans because the scenarios used in Fig. 9 correspond to: `TotalCoreLimit` = 1, `FcCoreCount` = 1, `FcBlock` = 500, `FcThermalizeCheck` = 500, and no spawning allowed. In essence, our one-dimensional scan along the $T_{\texttt{Low}}$ variable was done on a single core architecture. The absence of spawning accounts for a factor of a few in loss of boost, while the serial nature of the force call costs an additional factor of about 10. The fact that the optimum temperature depends on other parameters points to the high-dimensional and complex nature of our response surface. TAD can run at higher optimal $T_{\texttt{High}}$ values for more complex architectures because the parallelization of force calls and spawning of NEBs and transition checks can partially amortize the cost of processing the larger number of transitions that will occur before the stop time is reached.

OBSERVATION 2. *Spawning transition checks and NEBs is advantageous if enough cores are provided.*

Spawning of transition checks and NEBs are two key algorithmic variations for enhancing the basic TAD method. A comparison of TADSim with and without spawning is shown in Fig. 10 and Fig. 11. The parameters used for these scenarios are as follows: $T_{\texttt{Low}}$ = 200 K, `FcBlock` = 100, `FcThermalizeCheck` = 100, `FcCoreCount` = 64, `TotalCoreCount` = ∞. Fig. 10 shows the case where `TransCheckSpawnCoreCount` = `NEBBeadSpawnCoreCount` = `FcCoreCount` = 64. We see that spawning transition checks alone (green curve) increases the boost by about 25%. Spawning NEBs alone (blue curve) gives a larger effect ($\sim$ 60%), and the optimum T_{High} increases from 850 K to 1000 K, because the spawned NEBs hide most of the work caused by the larger number of attempted transitions at higher T_{High}. Once NEBs are spawned, spawning transition checks also (red curve) gives a larger effect than spawning transition checks alone. This is because the force-call savings represent a greater fraction of the time once the NEB cost has been eliminated. Note also that the optimum T_{High} then moves back down a bit, to 950 K.

Fig. 11 shows the same study using a minimal core count for the spawned work, `TransCheckSpawnCoreCount` = `NEBBeadSpawnCoreCount` = 1. The boost improvement from spawning the NEBs is still substantial; even `NEBBeadSpawnCoreCount` = 1 is effective since it provides one core for each of the 21 NEB beads, giving a net speedup of 21/9.9, and (more importantly) it offloads the work in parallel. In contrast, while the spawned single-core transition checks do improve the boost slightly at low values for T_{High}, they actually cause a loss of boost for T_{High} above 850 K, because it is quicker to do all the transition checks in serial with 64 cores (making each force call about 9.9x faster than a single core) than to wait for the slower single-core spawned transition check to complete once a transition does occur. This same detrimental behavior holds when both NEBs and transition checks are spawned, so that it is better to spawn just the NEBs in this case (blue curve vs. red curve). Note that the characteristics in both Fig. 10 and Fig. 11 will vary a bit with the choice of `FcBlock` and `FcThermalizeCheck`, which in this case were set to 100.

OBSERVATION 3. *Frequent transition checks improve performance in large core-count situations, while they are detrimental in limited-core situations.*

Fig. 12 shows the dependence on `FcBlock`, the inverse frequency of transition checks, using fixed parameters $T_{\texttt{Low}}$ = 300 K, `FcCoreCount` = 1 or 64, `TransCheckSpawnCoreCount` = 64, and `NEBBeadSpawnCoreCount` = 64. The value of `FcThermalizeCheck` was matched to `FcBlock`. We see that a serial code (red curve) gives its best performance when tran-

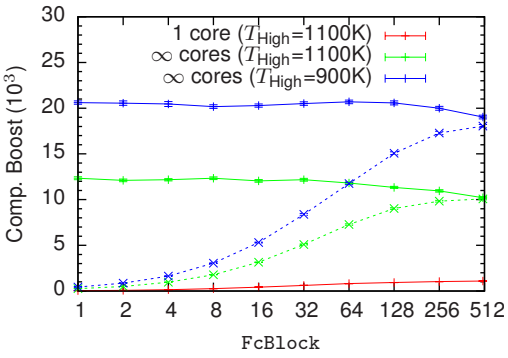

Fig. 12. Computational Boost as a function of the transition check interval `FcBlock` for spawned NEBs and transition checks (+) and if only NEBs are spawned (x).

sition checks are infrequent – there is a maximum at `FcBlock` = 512 (larger than 512 starts to degrade the physical predictions). In a parallel environment, at the same T_{High} = 1100 K, using `FcCoreCount` = 64 and spawning NEBs gives a substantial speedup (green curves at `FcBlock`=512). An additional 20% can be gained by shortening the time between transition checks (solid green curve), but only if they are spawned (solid green curve compared to dashed green curve), which requires that enough cores are available. As mentioned above, this performance gain stems from the fact that transitions can be detected sooner, which decreases the amount of "wasted", post-transition MD time. Looking at the T_{High} = 900 K case (blue curves), the effect is essentially the same, and we note again the importance of choosing an optimal high temperature. Finally, we note that this 20% boost enhancement due to shortening the time between transition checks should increase further if the MD time is accumulated in parallel, e.g., by using parallel-dynamics [Voter 1998], because the half-block post-transition time will be a greater fraction of the wall-clock time between transition events. We leave investigation of this effect for future work.

10. SUMMARY AND FUTURE PLANS

We have introduced the concept of a parameterized application simulator, TADSim, that models the TAD AMD method as a fast-running proxy using DES. This was accomplished by abstracting out the computationally intensive force calls and replacing them by the passage of time while keeping true to the execution flow of the original TAD application. This DES simulator results in a much lower computational cost than actual TAD runs for understanding and exploring performance-related behavior. New algorithmic extensions such as speculative spawning can be tested before being incorporated into the actual code. Parameter scans can be used to understand and explore the limits of the current TAD implementation.

TADSim was validated against the original TAD code using an example silver material system. Parameter scans using this same physical system have produced interesting results that have allowed us to gain insight into the TAD algorithm behavior and suggest performance enhancements. We have observed a clear optimum T_{High}, given a T_{Low}, where performance is best and we showed that T_{Low} and this optimal T_{High} come closer together with higher T_{Low}. We have evaluated the potential impact of the parallelization of TAD on performance. Adding parallelism to all force call computation is an effective way to achieve a direct speedup, assumed in the present case to give a 10-

fold effect. Spawning of transition checks and NEBs further improves the performance and shifts the optimal T_{High} to a higher temperature. Increasing the frequency of transition checks can improve performance when large core counts are available. Overall, the TAD algorithm can make use of hundreds of cores. These results add value for future method development and physics research in molecular dynamics.

We have seen that one of the important parameters in optimizing the TAD performance is the high temperature. A high value increases the potential speedup, while making it too high degrades performance by introducing excessive overhead. Shim and Amar have proposed ways [Shim and Amar 2011] to optimize the value for the high temperature on the fly in TAD simulations. Our work here complements and extends that work, in that we explore the dependence on a large number of parameters in addition to the high temperature, and we do so in the context of PDES.

Currently, TADSim supports exploration of the algorithmic choices more strongly than architectural choices. Processor speed, core counts for MD and spawning, and spawning start-up and tear-down timings allow for limited architectural exploration. Future plans include coupling a more robust hardware model representing nodes, memory, and communications. Parameters will be added to support exploration of power consumption and resilience. Dynamic power usage will be tracked based on a node's characteristics and computations being performed. The value of specific resilience strategies for detection and recovery on runtime and computation quality will be explored for given hardware error rates.

A ParRep application simulator has been developed that represents the parallel-replica dynamics AMD method [Voter 1998], publication forthcoming. Potential future directions include using TADSim for "on the fly" computational steering of TAD while running, exploring optimal AMD method combinations using plug-and-play component assemblies, and applying this DES approach to other application areas.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insights and helpful comments. The authors would also like to thank Kei Davis for his helpful comments. This research has been funded by the Los Alamos National Laboratory (LANL) LDRD program. The authors would like to acknowledge the Institutional Computing Program at LANL for use of their HPC cluster resources. C. J. acknowledges funding by a Los Alamos National Laboratory Director's fellowship. Assigned: Los Alamos Unclassified Report 13-28342. LANL is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of the U.S. DOE under Contract DE-AC52-06NA25396.

REFERENCES

Advanced Simulation and Computing Program. 2012. National Security Applications Co-Design Project. (2012). https://asc.llnl.gov/codesign/.

M. P. Allen and D. J. Tildesley. 1989. *Computer simulation of liquids*. Oxford University Press. 263 pages. Eq. (9.24).

P. Andelfinger and H. Hartenstein. 2013. Towards performance evaluation of conservative distributed discrete-event network simulations using second-order simulation. In *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS'13)*. ACM, 221–230.

R. Bagrodia, E. Deelman, S. Docy, and T. Phan. 1999. Performance Prediction of Large Parallel Applications Using Parallel Simulations. In *Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP'99)*. ACM, New York, NY, 151–162.

X.-M. Bai, A. F. Voter, R. G. Hoagland, M. Nastasi, and B. P. Uberuaga. 2010. Efficient annealing of radiation damage near grain boundaries via interstitial emission. *Science* 327, 5973 (2010), 1631–1634.

J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol. 2008. *Discrete-Event System Simulation*. Prentice Hall, Englewood Cliffs, NJ.

K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, and S. Pakin. 2009. Using performance modeling to design large-scale systems. *IEEE Computer* 42, 11 (2009), 42–49.

K. J. Barker, S. Pakin, and D. J. Kerbyson. 2006. A performance model of the Krak hydrodynamics application. In *Proceedings of the International Conference on Parallel Processing (ICPP'06)*. IEEE, Columbus, OH, 245–254.

G. Bauer, S. Gottlieb, and T. Hoefler. 2012. Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3_rmd. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'12)*. IEEE/ACM, Ottawa, Canada, 652–659.

N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The gem5 simulator. *Newsletter ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

A. B. Bortz, M. H. Kalos, and J. L. Lebowitz. 1975. A new algorithm for Monte Carlo simulation of Ising spin systems. *J. Comput. Phys.* 17, 1 (1975), 10–18.

R. N. Calheiros, M. A. S. Netto, C. A. F. De Rose, and R. Buyya. 2012. EMUSIM: Am Integrated Emulation and Simulation Environment for Modeling, Evaluation, and Validation of Performance of Cloud Computing Applications. *Softw. Pract. Exper.* 00 (2012), 1–18.

T. Clark. 1985. *A handbook of computational chemistry: A practical guide to chemical structure and energy calculations*. Wiley New York.

M. S. Daw and M. I. Baskes. 1984. Embedded atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B* 29 (1984), 6443–6453.

S. Eidenbenz, K. Davis, A. Voter, H. Djidjev, L. Gurvitz, C. Junghans, S. Mniszewski, D. Perez, N. Santhi, and S. Thulasidasan. 2012. Optimization Principles for Codesign Applied to Molecular Dynamics: Design Space Exploration, Performance Prediction and Optimization Strategies. *Proceedings of the U. S. Department of Energy Exascale Research Conference* (2012).

R. Ewald, J. Himmelspach, A. M. Uhrmacher, D. Chen, and G. K. Theodoropoulos. 2006. A simulation approach to facilitate parallel and distributed discrete-event simulator development. In *Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-time Applications (DS-RT'06)*. IEEE, 209–218.

R. M. Fujimoto. 1990. Parallel Discrete Event Simulation. *CACM* 33, 10 (1990), 30–53.

D. Gianni, G. Iazeolla, and A. D'Ambrogio. 2010. A methodology to predict the performance of distributed simulations. In *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS'10)*. IEEE Computer Society, 31–39.

G. Hendry and A. Rodriguez. 2012. Sst: A simulator for exascale co-design. In *Proceedings of the ASCR/ASC Exascale Research Conference*.

G. Henkelman and H. Jónsson. 2001. Long time scale kinetic Monte Carlo simulations without lattice approximation and predefined event table. *The Journal of Chemical Physics* 115 (2001), 9657.

Matthias Jeschke, Roland Ewald, and Adelinde M Uhrmacher. 2011. Exploring the performance of spatial stochastic simulation algorithms. *J. Comput. Phys.* 230, 7 (2011), 2562–2574.

H. Jónsson, G. Mills, and K. W. Jacobsen. 1998. Nudged elastic band method for finding minimum energy paths of transitions. In *CLassical and Quantum Dynamics in Condensed Phase Simulations*, Vol. 1. 385–404.

L. Kroc, S. Eidenbenz, and V. Ramaswamy. 2007. *SimCore*. Technical Report Los Alamos Unclassified Report 07-0590. Los Alamos National Laboratory, Los Alamos, NM.

Q. Li, Y. Dong, D. Perez, A. Martini, and R. W. Carpick. 2011. Speed dependence of atomic stick-slip friction in optimally matched experiments and molecular dynamics simulations. *Physical Review Letters* 106, 12 (2011), 126101.

J. Liu. 2010. Parallel Discrete-Event Simulation. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., New York, NY, 1–10.

R. Marcelin. 1915. *Ann. Physique* 3 (1915), 120.

J. A. McCammon and S. C. Harvey. 1988. *Dynamics of proteins and nucleic acids*. Cambridge University Press.

C. C. McGeoch. 2007. Experimental algorithmics. *Commun. ACM* 50, 11 (2007), 27–31.

C. C. McGeoch. 2012. *A guide to experimental algorithmics*. Cambridge University Press.

S. Mniszewski. 2010. *SimCore Tutorial*. Technical Report Los Alamos Unclassified Report 10-04282. Los Alamos National Laboratory, Los Alamos, NM.

Modeling & Networking Systems Research Group, Florida International University 2012. Parallel Real-time Immersive network Modeling Environment (PRIME). (2012). https://www.primessf.net/bin/view/Public/PRIMEProject.

D. Perez, B.P. Uberuaga, Y. Shim, J.G. Amar, and A. F. Voter. 2009. Accelerated Molecular Dynamics Methods: Introduction and Recent Developments. *Ann. Rep. Comp. Chem.* 5 (2009), 79.

S. Plimpton. 1995. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* 117, 1 (1995), 1–19.

D. C. Rapaport. 2004. *The art of molecular dynamics simulation*. Cambridge University Press.

A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, N. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. The structural simulation toolkit. *Newsletter ACM SIGMETRICS Performance Evaluation Review - Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems (PMBS 10)* 38, 4 (2011), 37–42.

N. Santhi, K. Davis, and S. Eidenbenz. 2013. *Graphical Models and a Scalable Methodology for Exascale Performance Prediction*. Technical Report Los Alamos Unclassified Report 13-24435.

Y. Shim and J. G. Amar. 2011. Adaptive temperature-accelerated dynamics. *J. Chem. Phys* 134 (2011), 054127.

M. R. Sorensen and A. F. Voter. 2000. Temperature-Accelerated Dynamics for Simulation of Infrequent Events. *J. Chem. Phys* 112, 21 (2000), 9599–9606.

K. L. Spafford and J. S. Vetter. 2012. ASPEN: A domain specific language for performance modeling. In *Supercomputing '12 (SC 2012)*. Salt Lake City, Utah.

M. O. Steinhauser and S. Hiermaier. 2009. A review of computational methods in materials science: examples from shock-wave and polymer physics. *International Journal of Molecular Sciences* 10, 12 (2009), 5135.

C. Tapus and J. Hickey. 2007. A Theory of Nested Speculative Execution. In *Proceedings of the 9th international conference on Coordination models and languages (Coordination'07)*. Springer-Verlag, Berlin, Heidelberg, 151–170.

U. S. Department of Energy Office of Science. 2012. Scientific Discovery through Advanced Computing (SciDAC) Co-Design. (2012). http://science.energy.gov/ascr/research/scidac/co-design/.

U. S. Department of Energy Office of Science. 2014. Applied Mathematics Research for Exascale Computing. (2014). http://science.energy.gov/ /media/ascr/pdf/research/am/docs/EMWGreport.pdf.

G. H. Vineyard. 1957. Frequency factors and isotope effects in solid state rate processes. *Journal of Physics and Chemistry of Solids* 3, 1 (1957), 121–127.

A. F. Voter. 1988. Simulation of the layer-growth dynamics in silver films: Dynamics of adatom and vacancy clusters on Ag (100). In *31st Annual Technical Symposium*. International Society for Optics and Photonics, 214–226.

A. F. Voter. 1994. The embedded atom method. *Intermetallic Compounds: Principles* 1 (1994), 77.

A. F. Voter. 1998. Parallel Replica Method for Dynamics of Infrequent Events. *Physical Review B* 57, xi224 (1998), 985–988.

A. F. Voter. 2007. Introduction to the kinetic Monte Carlo method. In *Radiation Effects in Solids*. Springer, 1–23.

A. F. Voter, F. Montalenti, and T.C. Germann. 2002. Extending the Time Scale in Atomistic Simulation of Materials. *Annu. Rev. Mater. Res.* 32 (2002), 321.

W. H. Wolf. 1994. Hardware-software co-design of embedded systems. *Proc. IEEE* 82, 7 (1994), 967–989.

H. Xia, H. Dail, H. Casonova, and A. Chien. 2004. The MicroGrid: Using Emulation to Predict Application Performance in Diverse Grid Network Environments. In *Proceedings of the 2nd International Workshop on Challenges of Large Applications in Distributed Environments (CLADE'04)*. IEEE, Honolulu, HI, 52–61.

G. Zheng, G. Kakulapati, and L. V. Kale. 2004. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE, Santa Fe, NM.